



Lisbon School  
of Economics  
& Management  
Universidade de Lisboa

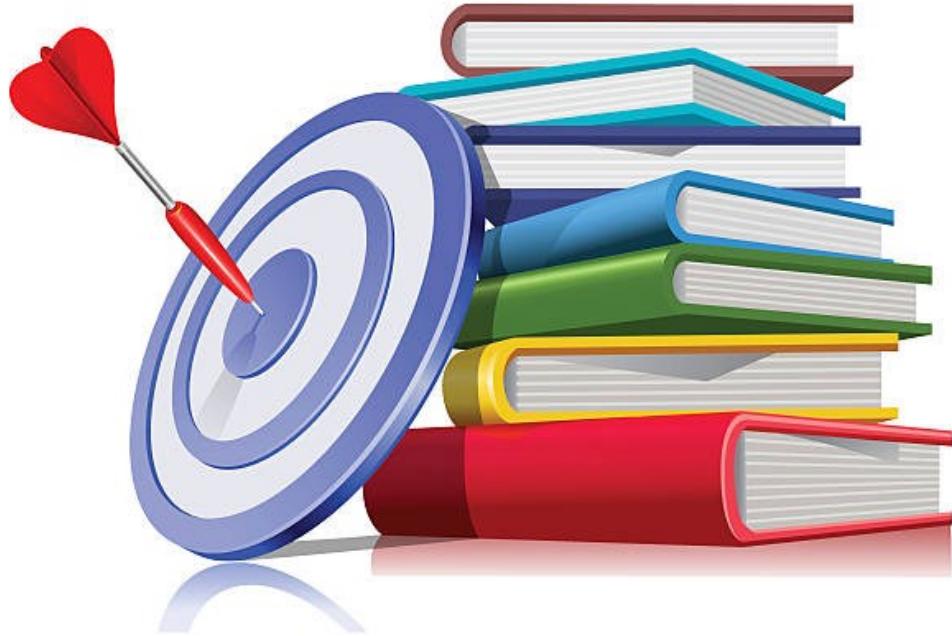


# Object Oriented Programming

Prof. Carlos J. Costa, PhD

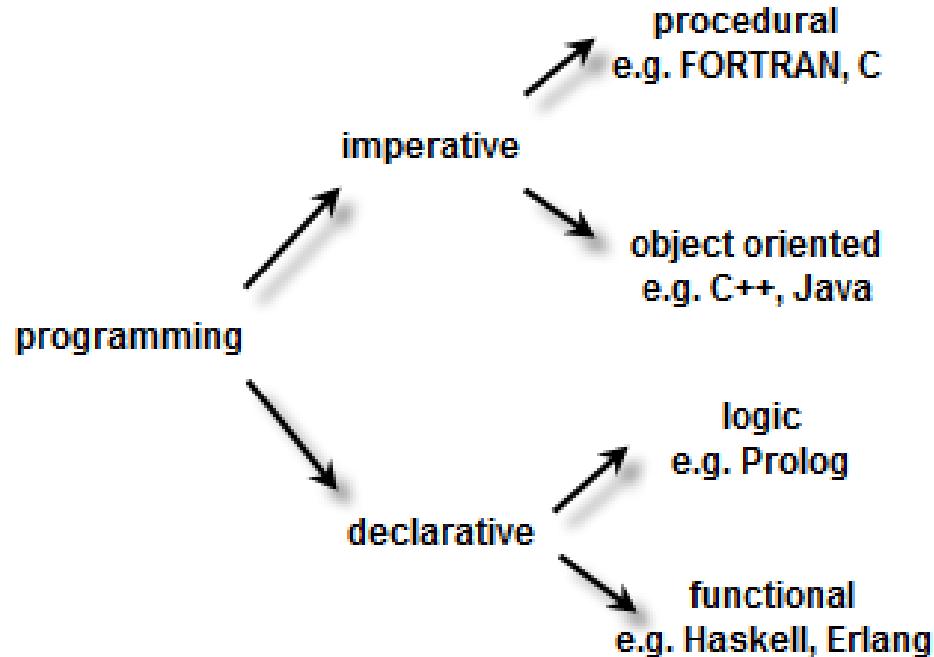
# Learning Objectives

- Understand the main concepts related to the object-oriented approach
- Understand how object-oriented programming is implemented in Python
- Create a small application with object-oriented programming



# Imperative Programming

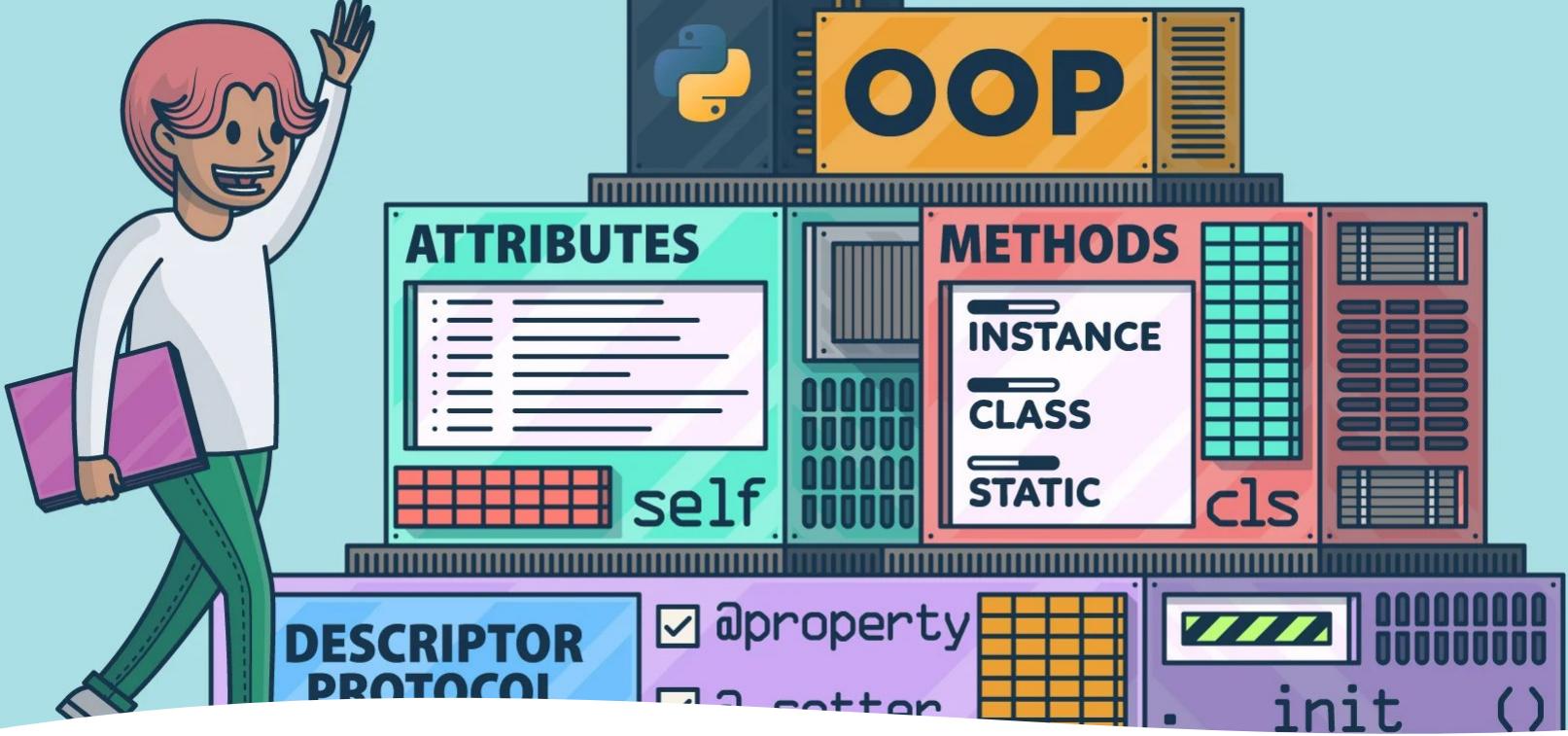
- Procedural - instructions grouped into procedures
- Object-Oriented - instructions grouped together with the part of the state they operate on.





# Object oriented Approach

- The main structural components of all systems are:
- Objects
  - Class Objects



# Object

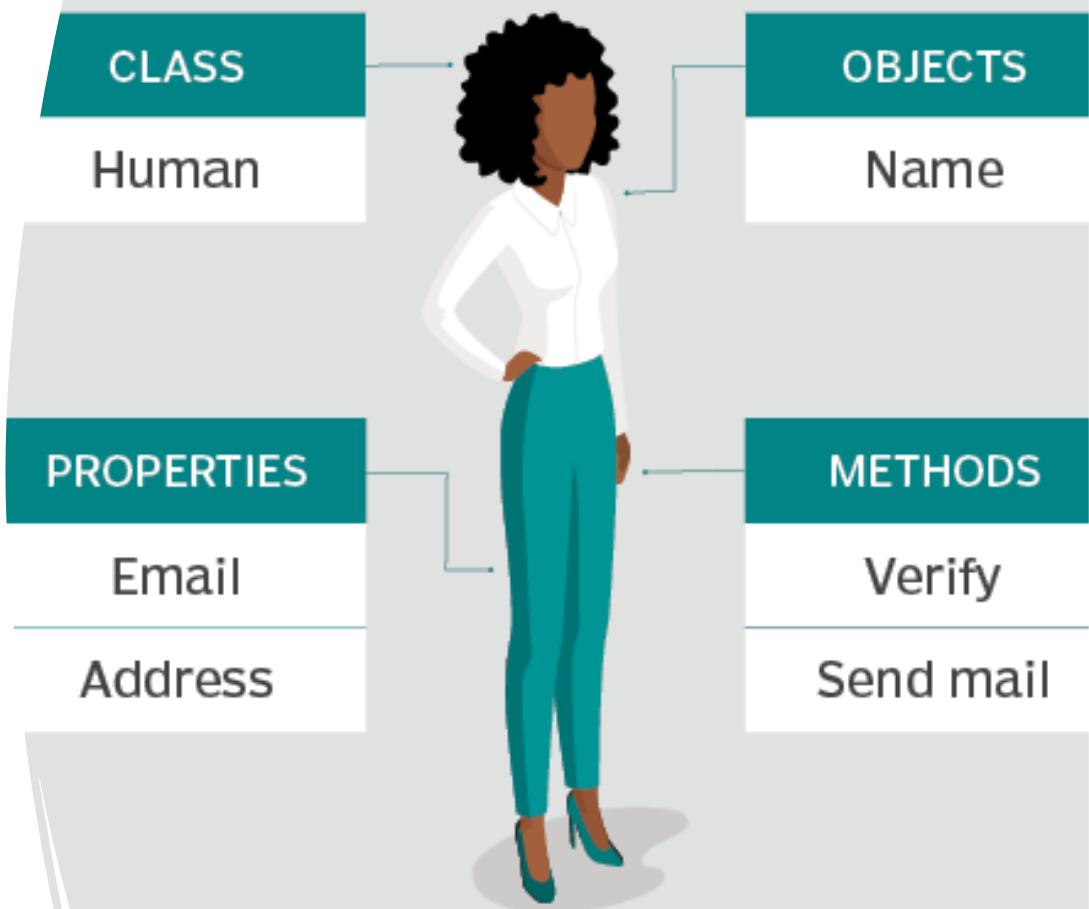
Object is something that takes up space in the real or conceptual world with which somebody may do things  
( Booch et al . 1999)

# Object-oriented programming

## Object

The objects have :

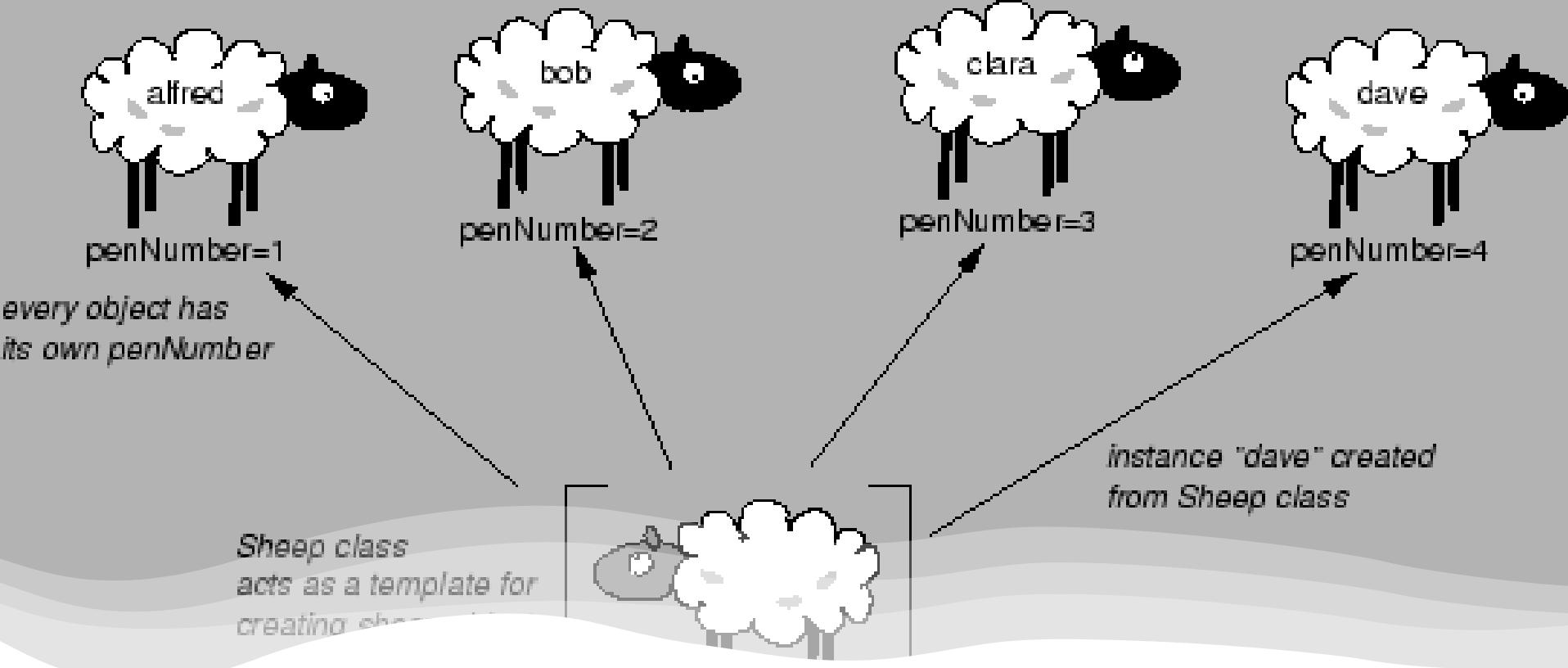
- Name (or ID )
- State
- Operations (or behavior )





Class is the blueprint of an object.

# Class

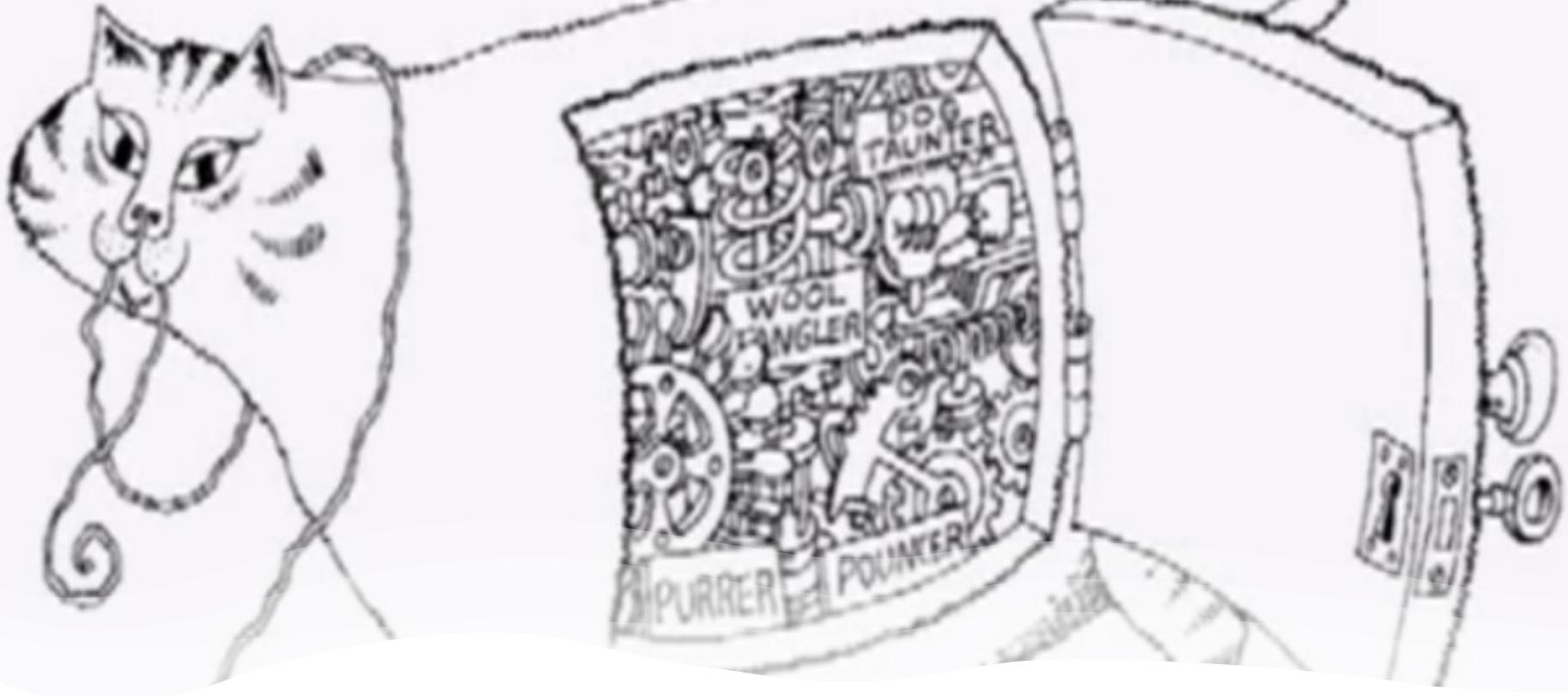


An object is an instance of a class.

## Instance

# Main characteristics of the approach

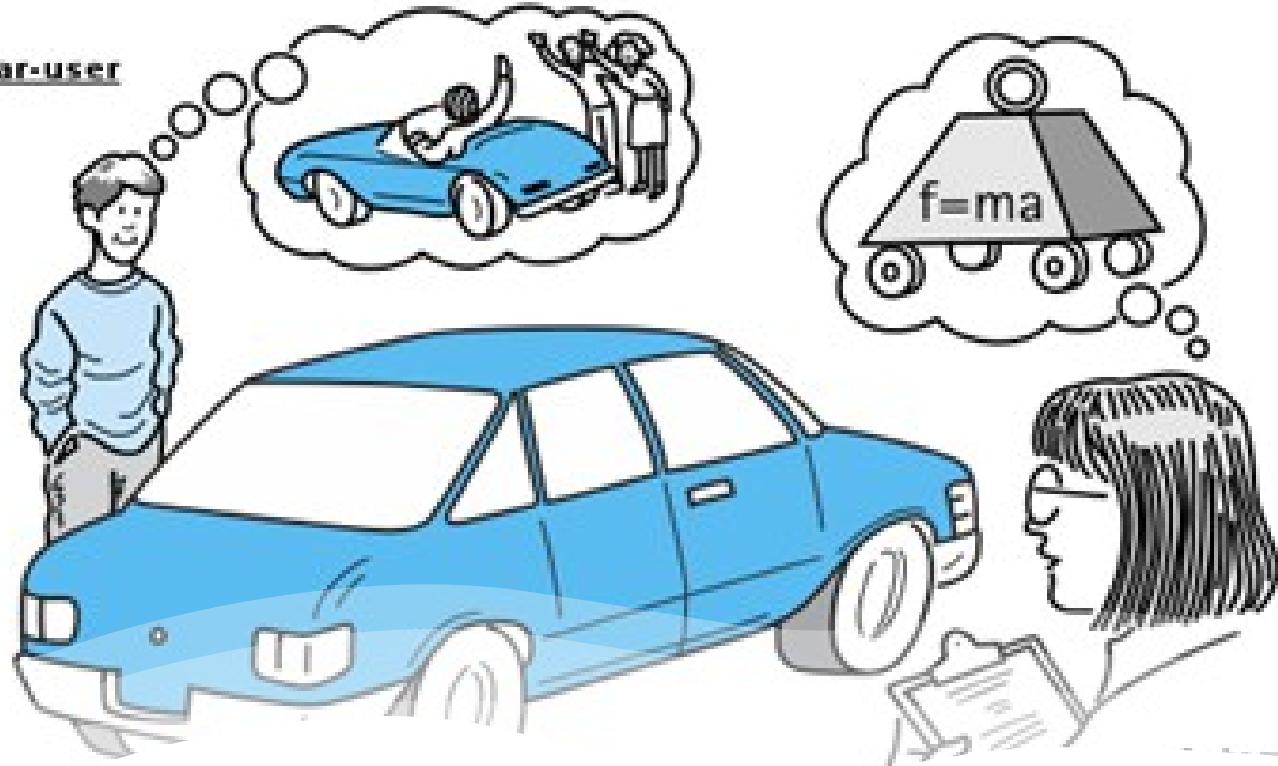
- encapsulation
- abstraction
- inheritance
- polymorphism



## Encapsulation

- is the mechanism of hiding the implementation of the object

Any car-user

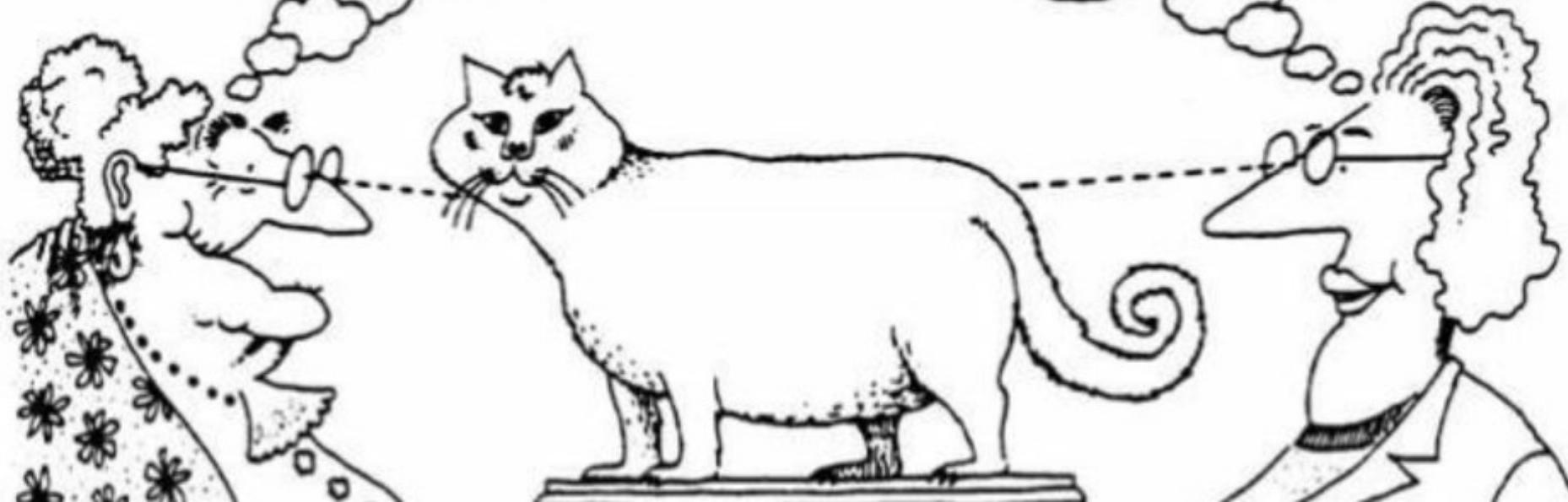
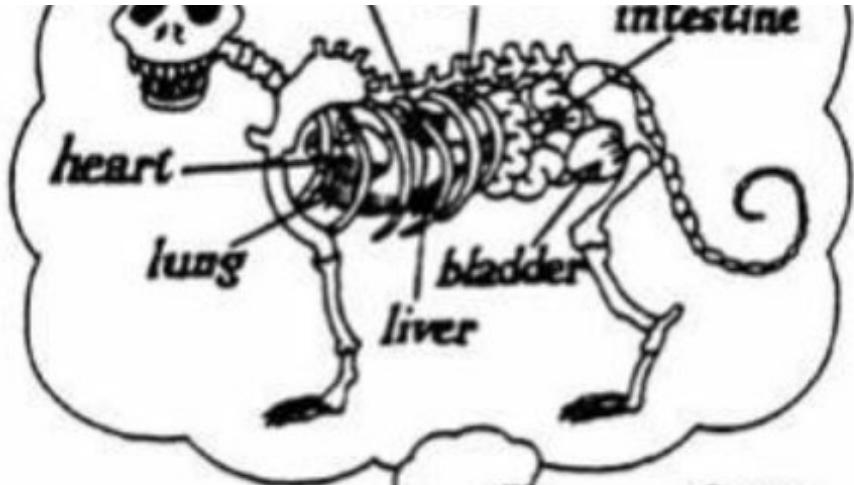


# Abstraction

- is a principle which consists of ignoring the aspects of a subject that is not relevant for the present purpose

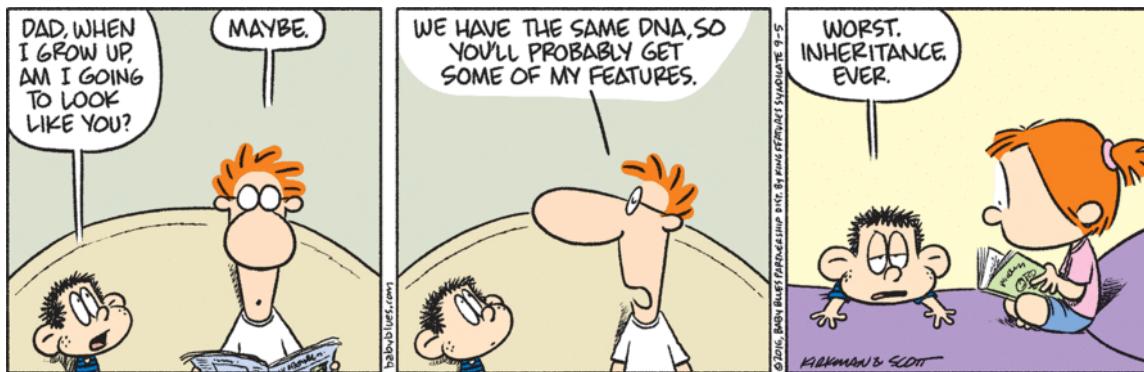
# Abstraction

Abstraction is the concise representation of a more complex object



# Inheritance

- Inheritance is the mechanism of making new classes from existing one.

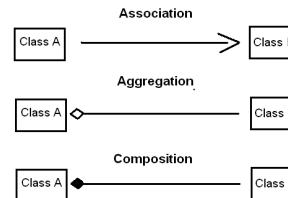
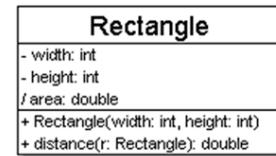


# Polymorphism

- The word polymorphism means having many forms.
- Same function name (but different signatures) being used for different types.

# Class Diagrams

- Elements of a class diagram :
  - Classes
  - Relations between classes
    - Associations
    - Compositions
    - Aggregations
    - Generalizations



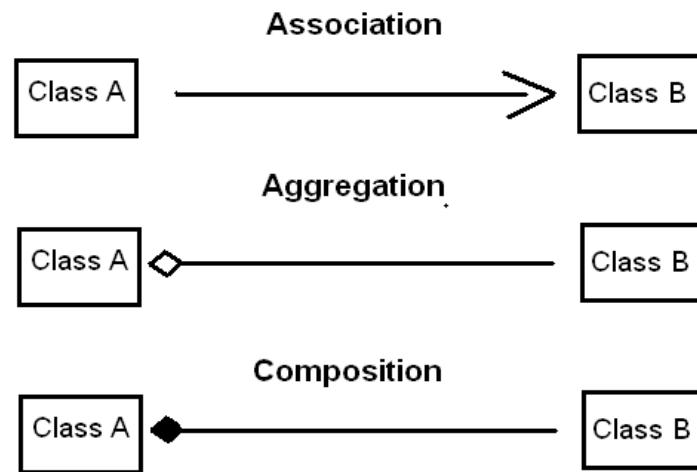
# Classe

Campaign
code
description
annual Cost
expected cost
pay()
do Budget()

- ID Class ( Class Name )
  - Refers to specific objects, but they must abstract
  - Nouns associated with the textual description of a problem
  - Choose carefully the names
  - using singular
- Attributes
  - Values that characterize the objects of a class
  - Types : Real, Integer , Text, Boolean , Enumerated , ...
- Operations
  - Behaviors of the class ( service, method)

# Relationship

- A relationship UML establishes the connection between elements

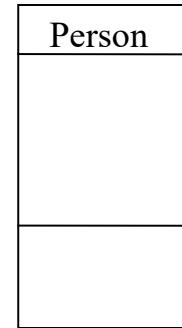


- Now let's go to



# Class

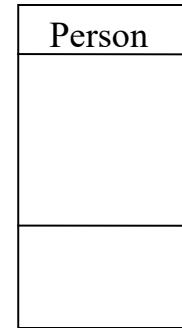
```
class Person:  
    pass # An empty block
```



# Class

```
class Person:  
    pass # An empty block
```

```
p = Person()  
print(p)
```



- **Result:**

```
<__main__.Person object at 0x0000021D9EED60F0>
```

# Method

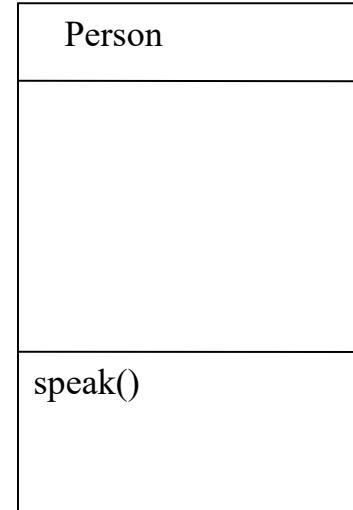
- Define class with method

class Person:

```
def speak(self):  
    print('Hello, how are you?')
```

- Create object and call method

```
p = Person()  
p.speak()
```

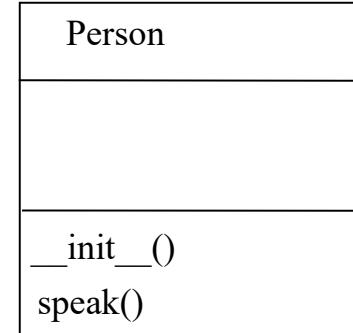


# **init** method

- The method **init()** is a special method,
- Is a method that Python calls when you create a new instance of this class.

# init method

```
class Person:  
    def __init__(self, name):  
        self.name = name  
    def speak(self):  
        print('Hello, my name is', self.name)  
p = Person('Carlos')  
p.speak()
```



# self

- The first argument of every class method, including `init`, is always a reference to the current instance of the class.
- By convention, this argument is always named `self`.

# Class Pet

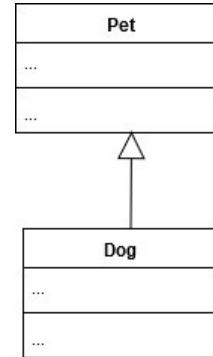
```
class Pet(object):  
    def __init__(self, name, species):  
        self.name = name  
        self.species = species  
    def getName(self):  
        return self.name  
    def getSpecies(self):  
        return self.species  
    def __str__(self):  
        return "%s is a %s" % (self.name, self.species)
```

# Inheritance

```
class Dog(Pet):
```

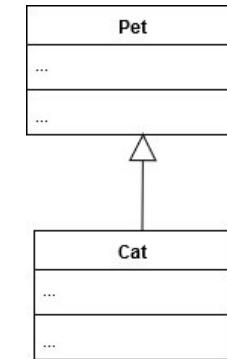
```
    def __init__(self, name, chases_cats):  
        Pet.__init__(self, name, "Dog")  
        self.chases_cats = chases_cats
```

```
    def chasesCats(self):  
        return self.chases_cats
```



# Inheritance

```
class Cat(Pet):  
    def __init__(self, name, hates_dogs):  
        Pet.__init__(self, name, "Cat")  
        self.hates_dogs = hates_dogs
```



```
def hatesDogs(self):  
    return self.hates_dogs
```

```
myPet = Pet("Boby", "Dog")
```

```
myDog = Dog("Boby", True)
```

```
isinstance(myDog, Pet)
```

```
isinstance(myDog, Dog)
```

```
isinstance(myPet, Pet)
```

```
isinstance(myPet, Dog)
```

# Access Modifiers

- Public,
- Private
- Protected

# Private

- They can be handled only from within the class.

class Person:

```
def __init__(self, name, age):  
    self.__name=name  
    self.__age=age
```

p=Person("David",23)

p.\_\_name

# Public

```
class Person:  
    def __init__(self, name, age):  
        self.name=name  
        self.age=age
```

```
p=Person("David",23)  
p.name
```

# Protected

```
class Person:  
    def __init__(self, name, age):  
        self._name=name  
        self._age=age
```

```
p=Person("David",23)  
p.name
```

# Class Person

```
class Person:
```

```
    def __init__(self, money=0, energy=100):
        self.money = money
        self.energy = energy
```

```
    def work(self, hours):
```

```
        if self.energy >= hours * 10:
```

```
            self.money += hours * 10 # Assume earning $10 per hour of work
```

```
            self.energy -= hours * 10
```

```
            print(f"Worked for {hours} hours. Money increased to ${self.money}. Energy decreased to {self.energy}.")
```

```
        else:
```

```
            print("Not enough energy to work.")
```

```
# Example usage:
```

# Adding Comments to Class

```
class Person:  
    """  
        Represents a person with attributes like money and energy.  
        Attributes:  
            money (int): The person's current amount of money.  
            energy (int): The person's current energy level.  
        Methods:  
            __init__(self, money=0, energy=100):  
                Initializes a new Person object with default values.  
            work(self, hours):  
                Simulates the person working for a certain number of hours,  
                earning money but losing energy.  
    """
```

# Adding comments to init method

```
class Person:  
    def __init__(self, money=0, energy=100):  
  
        """  
        Initializes a new Person object.  
        Args:  
            money (int, optional): The person's starting amount of money.  
                Defaults to 0.  
            energy (int, optional): The person's starting energy level.  
                Defaults to 100.  
        """  
  
        self.money = money  
        self.energy = energy
```

# Adding comments to methods

```
class Person:  
    ...  
    def work(self, hours):  
        """  
            Simulates the person working for a certain number of hours.  
            Args:  
                hours (int): The number of hours the person will work.  
  
            Raises:  
                ValueError: If the provided hours are negative.  
        """  
        if hours < 0:  
            raise ValueError("Hours cannot be negative.")  
  
        if self.energy >= hours * 10:  
            self.money += hours * 10  
            self.energy -= hours * 10  
            print(f"Worked for {hours} hours. Money increased to  
${self.money}. Energy decreased to {self.energy}.")  
        else:  
            print("Not enough energy to work.")
```

# Conclusions

- Object Oriented Approach
- Concept of Class, Object, Methods, Variables
- Inheritance and Modifiers access

# Bibliography

- Bennet, S. McRobb, S & Farmer, R., *Object Oriented Systems Analysis and Design using UML*, MacGarw-Hill, 1999.
- Booch, G., Rumbaugh, J. & Jacobson, I, *The Unified Modeling Language User Guide*. Addison Wesley, 1999 (tradução portuguesa brasileira \_\_\_\_; *UML Guia do Usuário*; Campus, 2000).
- Costa, C. *Desenvolvimento para Web*, ITML Press, 2007
- Nunes, M & O'Neill, H. *Fundamental de UML*, FCA, 2001
- Silva, A & Videira, C., *UML, Metodologias e Ferramentas CASE*, Edições Centro Atlântico, 2001
- Terry, Q. *Visual Modeling With Rational Rose 2000 and UML*, Addison-Wesley. 2000.
- *Oxford Dictionary of Computing*, Oxford University Press.